Documentation: Surya’s Text to Math Problem Solver and Data Research Assistant Using Streamlit

Overview

This project is a **Streamlit-based chatbot** that assists users in solving mathematical problems step-by-step and retrieving relevant data from Wikipedia. The chatbot uses **Groq's AI models** to process user queries and provides detailed explanations for mathematical solutions.

Features

1. Text-to-Math Problem Solving

- Accepts a **math problem** from the user.
- Converts the problem into a **numerical expression** and solves it.
- Provides **step-by-step explanations** for the solution.

2. Wikipedia Data Retrieval

- Uses **Wikipedia API** to fetch relevant information.
- Helps with **concept explanations** related to math problems.

3. Intelligent Agent for Answering Queries

- Uses **LLM-based tools** to process questions.
- Can handle **complex problem-solving tasks** with logical reasoning.

Architecture & Components

1. Streamlit UI

- **Sidebar** for entering the **Groq API Key**.

- **Main interface** to enter a math question and view responses.
- Displays **chat history** with user and assistant messages.

2. Language Model (LLM) - ChatGroq

- The system connects to **Groq's AI models** for generating responses.
- The model used is **"gemma2-9b-it"**.

3. Wikipedia API Wrapper

- Enables the assistant to **search Wikipedia** for additional knowledge.
- Used as a tool to **supplement answers** when needed.

4. Math Processing System

- Uses an **LLM-based Math Chain** to evaluate **numerical expressions**.
- A custom **prompt template** is designed to ensure valid calculations.
- Extracts the **mathematical expression** from the user's question.
- Handles cases where **variables** need to be assigned values.

5. Reasoning System

- A dedicated tool for **step-by-step logical reasoning**.
- Breaks down the solution into a **structured explanation**.
- Uses a **custom prompt** for structured reasoning responses.

6. AI Agent Integration

- Combines multiple tools (**Wikipedia, Math Solver, Reasoning**) into a **single intelligent assistant**.
- Uses **Zero-Shot ReAct Agent** for decision-making.
- Handles **parsing errors** gracefully.

How It Works

1. The user enters a **math problem** in the text box.
2. The chatbot determines whether to:
    - **Calculate directly** (if it's a straightforward math question).
    - **Use logical reasoning** (if explanation is needed).

    - **Fetch data from Wikipedia** (if additional knowledge is required).
3. The response is displayed with:
    - **Numerical expression** extracted from the question.
    - **Step-by-step explanation** of the solution.
    - **Final result** with detailed insights.

Error Handling & Robustness

- **Validates API Key** before running the chatbot.
- **Ignores invalid mathematical expressions** to prevent errors.
- **Handles missing numerical values** by making reasonable assumptions.
- **Filters Wikipedia results** to return only relevant information.
- **Displays error messages** if something goes wrong.

User Interaction Flow

1. User **enters a math problem**.
2. The assistant **analyzes the question**.
3. If it's a **direct calculation**, the **math solver** is used.
4. If **reasoning is required**, the **reasoning tool** generates a structured explanation.
5. If **additional information is needed**, the **Wikipedia tool** fetches related concepts.
6. The response is displayed in a **chat-style format**.

Possible Use Cases

- **Students** learning math who need step-by-step explanations.
- **Teachers & tutors** who want to verify and explain math solutions.
- **Researchers** looking for quick Wikipedia-based references.
- **Casual users** who need help solving math problems.

Limitations

- Requires a **valid Groq API Key** to function.
- Assumptions for **unknown variables** may not always match user expectations.
- Wikipedia tool is **limited to English-language articles**.
- May not handle **highly advanced symbolic algebra**.

Future Enhancements

- Integrating a **graphing feature** for visual explanations.
- Supporting **LaTeX rendering** for math expressions.
- Enhancing Wikipedia search with **multi-source document retrieval**.
- Adding **voice input** for a more interactive experience.

Conclusion

This **AI-powered assistant** simplifies complex math problems and provides **detailed, step-by-step solutions**. It integrates **Wikipedia search** and **logical reasoning** to enhance learning and research. Built with **Streamlit**, this tool makes math problem-solving **intuitive and user-friendly**.

Code:

```
import streamlit as st

from langchain_groq import ChatGroq

from langchain.chains import LLMMathChain, LLMChain

from langchain.prompts import PromptTemplate

from langchain_community.utilities import WikipediaAPIWrapper

from langchain.agents.agent_types import AgentType

from langchain.agents import Tool, initialize_agent
from dotenv import load_dotenv

from langchain.callbacks import StreamlitCallbackHandler

import re

# Load environment variables

load_dotenv()

# Setting up the Streamlit app

st.set_page_config(page_title="Surya's Text to Math Problem Solver and Data Research Assistant")

st.title("Text to Math Problem Solver")

# Get Groq API Key from sidebar

groq_api_key = st.sidebar.text_input(label="Groq API Key", type="password")

if not groq_api_key:

   st.info("Please type your GROQ API key to continue")

   st.stop()

# Initialize the language model

try:

   llm = ChatGroq(model="gemma2-9b-it", groq_api_key=groq_api_key)

except Exception as e:

   st.error(f"Failed to initialize model: {e}")

   st.stop()

# Initialize the Chat Tools
```

```
wikipedia_wrapper = WikipediaAPIWrapper()

wikipedia_tool = Tool(

  name="Wikipedia",

  func=wikipedia_wrapper.run,

  description="A tool for searching Wikipedia to assist with math problems"

)

# Initialize the Math tool with enhanced explanation

try:

  math_chain = LLMMathChain.from_llm(llm=llm)

  # Define a custom math prompt to enforce numerical expressions

  math_prompt = """

  You are a mathematical assistant. For the given question, provide a numerical expression (no
variables) and a detailed, point-wise explanation of how to solve it. If the question contains
variables, assume reasonable numerical values (e.g., k=1) and state your assumption:

  Question: {question}

  Numerical Expression: <expression>

  Description:

  - Step 1: [First step]

  - Step 2: [Second step]

  - ... [Continue as needed]
```

```
Final Result: <result>

"""

math_prompt_template = PromptTemplate(input_variables=["question"], template=math_prompt)

math_explain_chain = LLMChain(llm=llm, prompt=math_prompt_template)

def calculate_with_explanation(question):

    try:

        # Get explanation from LLM

        explanation = math_explain_chain.run({"question": question})

        # Extract numerical expression

        expr_match = re.search(r'Numerical Expression: (.*?)\n', explanation)

        if not expr_match:

            return f"Question: {question}\nError: No valid numerical expression
provided\nDescription:\n- Step 1: Failed to parse a numerical expression\nFinal Result: N/A"

        expr = expr_match.group(1).strip()

        # Check for variables and handle them

        if re.search(r'[a-zA-Z]', expr):

            assumption = "Assumption: Any variables (e.g., k) set to 1 unless specified."

            expr = re.sub(r'[a-zA-Z]', '1', expr)  # Replace variables with 1

            explanation = f"{explanation}\n{assumption}"

        result = math_chain.run(expr)
```

```
            return f"{explanation}\nFinal Result: {result}"

        except Exception as e:

            return f"Question: {question}\nNumerical Expression: {expr if 'expr' in locals() else
'N/A'}\nDescription:\n- Error: {e}\nFinal Result: N/A"

    calculator = Tool(

        name="Calculator",

        func=calculate_with_explanation,

        description="Solves math questions with step-by-step explanations."

    )

except Exception as e:

    st.error(f"Math tool initialization failed: {e}")

    st.stop()

# Define the reasoning prompt with step-by-step description

prompt = """

You are an agent tasked with solving the user's mathematical question. Provide a numerical
expression (no variables) and a detailed, point-wise explanation. If variables are present, assume
reasonable values (e.g., k=1) and note the assumption:

Question: {question}

Numerical Expression: <expression>

Description:

- Step 1: [First step]
- Step 2: [Second step]

- ... [Continue as needed]

Final Result: <result>

"""

prompt_template = PromptTemplate(input_variables=["question"], template=prompt)

# Combine tools into a chain

try:

   chain = LLMChain(llm=llm, prompt=prompt_template)

   reasoning_tool = Tool(

       name="Reasoning",

       func=chain.run,

       description="Answers math questions with step-by-step explanations."

   )

except Exception as e:

   st.error(f"Reasoning tool setup failed: {e}")

   st.stop()

# Initialize the agent

try:

   assistant_agent = initialize_agent(

       tools=[wikipedia_tool, calculator, reasoning_tool],
       llm=llm,
       agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
       verbose=False,
       handle_parsing_errors=True,
   )
except Exception as e:
   st.error(f"Agent initialization failed: {e}")
   st.stop()
# Initialize session state for messages
if "messages" not in st.session_state:
   st.session_state["messages"] = [
       {"role": "assistant", "content": "Hi, I'm a Math Chatbot who can solve your math questions with
step-by-step explanations!"}
   ]
# Display chat history
for msg in st.session_state.messages:
   st.chat_message(msg["role"]).write(msg["content"])
# Function to generate the response
def generate_response(user_question):
   try:
       response = assistant_agent.invoke({'input': user_question})

       resp = response['output'] if isinstance(response, dict) and 'output' in response else str(response)

       if "Description:" not in resp:

           return calculate_with_explanation(user_question)  # Fallback for steps

       return resp

   except Exception as e:

       return f"Error: {e}"

# Interaction logic

question = st.text_area("Enter your Question:")

if st.button("Find my answer"):

   if question and question.strip():

       with st.spinner("Generating Response..."):

           st.session_state.messages.append({"role": "user", "content": question})

           st.chat_message("user").write(question)

           st_cb = StreamlitCallbackHandler(st.container(), expand_new_thoughts=False)

           response = generate_response(question)

           st.session_state.messages.append({"role": "assistant", "content": response})

           st.write("### Response:")

           st.success(response)

   else:
```

```
    st.warning("Please enter a question")
```